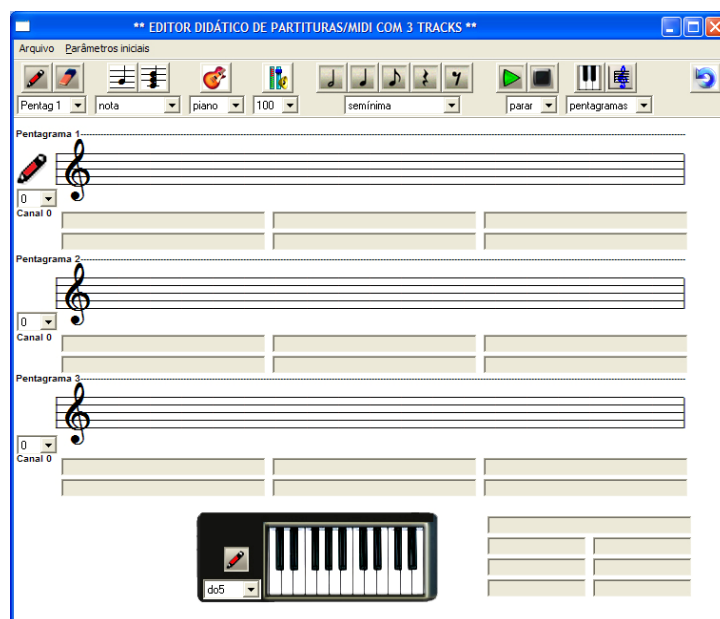


Capítulo 5

Editor Didático de Partituras.

O Anexo X apresenta provas de conceito de como criar interfaces visuais multimídia, informações necessárias para quem não domina esta técnica utilizando linguagem funcional pura (no caso, Clean). O programa a seguir mostra como, a partir das provas de conceitos e manipulação do protocolo MIDI, implementar um editor de partituras em linguagem funcional pura contendo os recursos e conceitos mais relevantes utilizados em editores profissionais de partitura, inclusive apresentando novas soluções não destrutivas inexistentes nos mesmos. Este aplicativo engloba praticamente todos os conceitos e exemplos registrados nos Anexos e no conhecimento registrado nos capítulos anteriores, tanto os de interface visual quanto os de programação e tipos únicos. Um estudo de casos de uso deste editor será mostrado no Capítulo 6, ficando, aqui, apenas os detalhes de implementação.



Interface do editor didático de partituras

5.1 Controle interativo variável de popup através de ícones da barra de ferramentas.

Foi utilizado neste editor o controle de seleção de item de popups através de um ícone da barra de ferramentas, onde, a cada clique, um novo item é selecionado em sequência, tais como:

- o popup de escolha de status nota/ acorde;
- o popup de instrumento;
- o popup de volume;
- o popup de figuras musicais;
- o popup de nota;
- o popup de escolha de teclado virtual/pentagrama.

A seguir é mostrada a lógica de seleção de cada um:

5.1.1 Status nota/ acorde.



Figura 5.1 – “Toolbar” Nota / Acorde

O problema neste controle é que se deve criar na lista de status se a nota colocada é nota solo ou uma nota pertencente a um acorde. Outro problema é indicar que se estando em um acorde, vai se iniciar um novo acorde. Assim, uma nota solo tem o código 1, um acorde o código 0 e um novo acorde, a partir de um acorde ou quando se apaga uma nota entre dois acordes, possui o código 2.

Se o ícone de nota da barra de ferramentas for pressionado, ele chama a função **botaoNota** que coloca o status de **nota** (código 1) na nota e ativa o item nota no menu popup de status nota/ acorde.

```
2546 botaoNota ids proc
2547 # proc = escreve ids proc
2548 # proc = appPIO (selectPopUpControlItem (ids!!84) 1 ) proc
2549 = proc
```

Listagem 5.1 – função “botaoNota”

A função *botaoNota* chama a função *escreve* que lê do estado local do processo o registro onde indica o pentagrama em que se deverá escrever a nota, e coloca o status do popup (identificador 84) como nota (**selectPopUpControlItem (ids!!84) 1**).

Se o ícone de acorde for pressionado, ele chama a função *botaoAcorde* que coloca o status de acorde (código 0 ou 2) na nota e ativa o item *acorde* (código 0) ou *novo acorde* (código 2) no menu popup de status nota/acorde.

```

2553 botaoAcorde ids proc=: {ls={tipoAcorde}}
2554 # proc = escreve ids proc
2555 # (Just dial, proc) = accPIO(getWindow (ids!!93)) proc
2556 # (se,Just tipo) = getControlText (ids!!84) dial
2557 | (tipoAcorde=="acorde")
2558   # proc = appPIO (selectPopUpControlItem (ids!!84) 3 ) proc
2559   = {ls = {proc.ls& tipoAcorde="novo acorde"}, io = proc.io }
2560 | otherwise
2561   # proc = appPIO (selectPopUpControlItem (ids!!84) 2 ) proc
2562   = {ls = {proc.ls& tipoAcorde="acorde"}, io = proc.io }
2563

```

Listagem 5.2 – Função “botaoAcorde”

A função *botaoAcorde* chama a função *escreve* que lê do estado local do processo o registro onde indica o pentagrama em que se deverá escrever a nota, e lê no estado local do processo o tipo de *status* que foi utilizado pela última nota (tipoAcorde). Se for nota coloca o status de acorde (código 0) (**selectPopUpControlItem (ids!!84) 2**), se for acorde coloca o status de novo acorde (código 2) (**selectPopUpControlItem (ids!!84) 3**), e, se for novo acorde, coloca o status de acorde (código 0) (**selectPopUpControlItem (ids!!84) 2**) no popup (identificador 84).

5.1.2 Escolha de instrumento.



Figura 5.2 – “Toolbar” Instrumento

Ao se pressionar o botão de escolha de instrumento, a função *botaoInstrumento* é ativada. A cada vez que o botão é pressionado, o item seguinte do popup de instrumento (**ids!!83**) é selecionado. Ao chegar no último instrumento, item 4,

inicia-se a contagem em 1 (instrumento flauta). Assim, a função **botaoInstrumento** lê o registro no estado local do processo e verifica qual é a contagem atual (**contagemInstr**) e segue a regra já descrita. Ao sair da função, atualiza-se o registro com o novo valor da contagem(**ls = {proc.ls& contagemInstr=contagem}**).

```

2566 botaoInstrumento ids proc=: {ls={contagemInstr}}
2567 | (contagemInstr == 4)
2568   # proc = appPIO (selectPopUpControlItem (ids!!83) 1 ) proc
2569   = {ls = {proc.ls& contagemInstr=1}, io = proc.io }
2570 # contagem = contagemInstr + 1
2571 # proc = appPIO (selectPopUpControlItem (ids!!83) contagem ) proc
2572 = {ls = {proc.ls& contagemInstr=contagem}, io = proc.io }
2573

```

Listagem 5.3 – função “botaoInstrumento”

5.1.3 Volume.

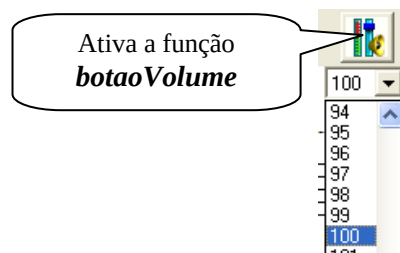


Figura 5.3 – “Toolbar” Volume

Ao se clicar no ícone de volume (um slider, um VU e um altofalante), a função **botaoVolume** é ativada. O princípio de atualização do popup de volume é similar ao de instrumento, o índice é escolhido com valores discretos de índice de 51 em 51 contagens, começando em 0. Assim, tem-se os volumes: 0, 50, 100 e 127. Quando a contagem atinge um valor superior a 127 (seria 150), a contagem atual passa a ser a de índice 127 e o registro do estado local do processo (**volum**) passa a assumir o primeiro índice com valor 0 (a contagem vai de 0 a 127). Para que esta sequência de volumes seja a escolhida, quando a função termina, o registro é atualizado com **volum** igual à contagem atual – uma unidade (**ls = {proc.ls& volum=(contagem-1)}**).

```
2576 botaoVolume ids proc=: {ls={volum}}
2577 | (volum == 127)
2578   # proc = appPIO (selectPopUpControlItem (ids!!82) 1 ) proc
2579   = {ls = {proc.ls& volum=0}, io = proc.io }
2580 # contagem = volum + 51
2581 | (volum == 100)
2582   # proc = appPIO (selectPopUpControlItem (ids!!82) 128 ) proc
2583   = {ls = {proc.ls& volum=127}, io = proc.io }
2584 # proc = appPIO (selectPopUpControlItem (ids!!82) contagem ) proc
2585 = {ls = {proc.ls& volum=(contagem-1)}, io = proc.io }
```

Listagem 5.4 – Função “botaoVolume”

5.1.4 Figuras musicais.



Figura 5.4 – “Toolbar” Figuras musicais

A lógica de atualização das figuras musicais é diferente da lógica de volume e semelhante, em parte, à de status nota/acorde. Neste caso, ao se clicar em um dos cinco ícones com figuras musicais, a cada botão está associado uma função, e cada função seleciona a figura musical correspondente no popup de figuras (**identificador 81**). Assim, a mínima é item 1, semínima é item 2, colcheia é item 3, pausa de semínima é item 4 e a pausa de colcheia é item 5.


```

2936 atualizar ids proc
2937     # proc = appPIO (setWindowLook (ids!!93) True (False, lookPentagrama)) proc
2938     #proc = atualizaTrack1 ids  proc
2939     #proc = atualizaTrack2 ids  proc
2940     #proc = atualizaTrack3 ids  proc
2941     =proc

```

Listagem 5.6 – Função “atualizar”

5.3 Plotar nota musical.

Para se plotar uma nota musical a ação é similar à de se *plotar* um pentagrama e a clave. Basta escolher uma fonte contendo as notas musicais, determinar o tamanho da letra e *plotar* a mesma na região correta do pentagrama. Para tanto, descreve-se, para cada nota, a coordenada *y* que a mesma deverá ser *plotada*. A coordenada *x* é variável de acordo com o local aonde vai se *plotar* a nota. Assim, estando no modo escrita, ao se clicar em uma tecla do teclado virtual, uma nota musical é escolhida no popup. Feito isto a função de *plotar* notas é chamada:

proc = montaTrack nota proc

Depois disto, a função de plotar nota, **montaTrack**, checa em qual pentagrama deverá plotar

|(pentag == "Pentag 1")

Feito isto, a coordenada *x* é incrementada em 50 *pixels* (cada nota possuirá uma distância de 50 *pixels* uma da outra)

incrementa1 = coordenadaX1 + 50

De posse das informações da nota e posição *x y*, a função de escrita em pentagrama (**look**) é chamada

```

# proc    = appPIO (setWindowLook wd1 True (False, (look "Petrucchi" 22 figura nota incrementa1 0 cor notaAcorde)))
proc

```

Onde o formato de *look* é:

look letra tamanho figura nota coordenadaX coordenadaY corNota nota/acorde

Assim, dependendo de qual pentagrama é, o valor de *y* muda. Se o status for de nota, a coordenada *X* é incrementada, se for acorde, a coordenada *X* é mantida, registrada no estado local do processo, e somente mudará quando o status nota/acorde mudar.

5.4 Apagar nota musical .

Para apagar uma nota, deve-se mudar o modo de edição para borracha. A escolha do modo de edição é realizada através da mudança de um campo do registro do estado local do processo (**tipoMouse**), ao se clicar no ícone lápis ou borracha. A escolha de qual pentagrama será editado também é feita através de outro campo do registro do estado local do processo (**pentag**). A figura 5.6 mostra na interface os controles que definem o modo de edição e o pentagrama que será ativado para edição.

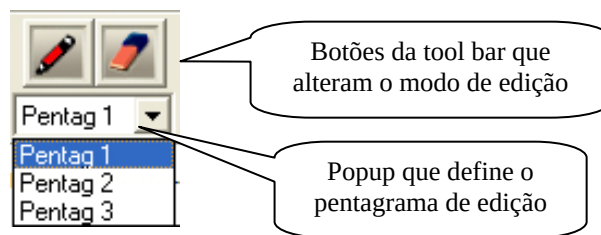


Figura 5.6 – Escolha do modo de edição e pentagrama

A listagem 5.7, a seguir, mostra o trecho de código de programa que define o modo de escrita e de apagar, definindo, também o pentagrama de trabalho e as mudanças na interface, tais como o ícone do modo de edição (lápis ou borracha), o teclado virtual, e outros parâmetros do registro do estado local do processo que serão utilizados pelas funções em cada modo.

```

2606 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2607 escreve ids proc=: {ls={tipoMouse,normal,tecladoEsc,branco,pentag}}// lapis
2608 # (Just dial, proc) = accPIO(getWindow (ids!!93)) proc
2609 # (se,Just pentg) = getControlText (ids!!85) dial
2610 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=221,y=473} tecladoEsc)) proc
2611 # proc = appPIO (setControlText (ids!!58) "") proc
2612
2613 |(pentg=="Pentag 1")
2614 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=60} normal)) proc
2615 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} branco)) proc
2616 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} branco)) proc
2617 = {ls = {proc.ls& tipoMouse="normal.bmp",pentag=pentg}, io = proc.io }
2618 |(pentg=="Pentag 2")
2619 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} normal)) proc
2620 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=60} branco)) proc
2621 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} branco)) proc
2622 = {ls = {proc.ls& tipoMouse="normal.bmp",pentag=pentg}, io = proc.io }
2623 |(pentg=="Pentag 3")
2624 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} normal)) proc
2625 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=60} branco)) proc
2626 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} branco)) proc
2627 = {ls = {proc.ls& tipoMouse="normal.bmp",pentag=pentg}, io = proc.io }
2628 = {ls = {proc.ls& tipoMouse="normal.bmp",pentag=pentg}, io = proc.io }
2629
2630 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2631 apaga ids proc=: {ls={tipoMouse,borracha,branco,preto,pentag}}// borraCHA
2632 # (Just dial, proc) = accPIO(getWindow (ids!!93)) proc
2633 # (se,Just pentg) = getControlText (ids!!85) dial
2634 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=218,y=470} preto)) proc
2635 |(pentg=="Pentag 1")
2636 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=60} borracha)) proc
2637 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} branco)) proc
2638 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} branco)) proc
2639 = {ls = {proc.ls& tipoMouse="borracha.bmp",pentag=pentg}, io = proc.io }
2640 |(pentg=="Pentag 2")
2641 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} borracha)) proc
2642 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=60} branco)) proc
2643 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} branco)) proc
2644 = {ls = {proc.ls& tipoMouse="borracha.bmp",pentag=pentg}, io = proc.io }
2645 |(pentg=="Pentag 3")
2646 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} borracha)) proc
2647 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=60} branco)) proc
2648 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} branco)) proc
2649 = {ls = {proc.ls& tipoMouse="borracha.bmp",pentag=pentg}, io = proc.io }
2650 = {ls = {proc.ls& tipoMouse="borracha.bmp",pentag=pentg}, io = proc.io }
2651
2652 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

Listagem 5.7 – Modos: escrita e apagar

Este trecho do programa mostra o processo de ajuste da interface e parâmetros nos modos escrita e apagar, para tanto, inicia pela leitura do registro do estado local, seja no modo escrita ou apagar, como segue nas listagens 5.8 e 5.9:

```

2607 escreve ids proc=: {ls={tipoMouse,normal,tecladoEsc,branco,pentag}}.

```

Listagem 5.8 – Leitura do registro, modo escrita

```

2631 apaga ids proc=: {ls={tipoMouse,borracha,branco,preto,pentag}}.

```

Listagem 5.9 – Leitura do registro, modo apagar

Logo após a leitura do registro, o sistema atualiza no teclado virtual o modo de edição. Se for escrita, coloca um lápis no teclado virtual, caso contrário, deixa a área do ícone em preto, conforme listagens 5.10 e 5.11.

```

2610 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=221,y=473} tecladoEsc)) proc

```

Listagem 5.10 – Plotagem de lápis no teclado virtual – modo leitura

```

2634 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=218,y=470} preto)) proc

```

Listagem 5.11 – Plotagem de bitmap preto no teclado virtual – modo apagar

Feita a leitura do popup que define o pentagrama no qual se efetuará a edição, pela linha de programa mostrada na listagem 5.12, a seguir, pode-se configurar a interface e atualizar o registro do estado local do processo para o modo escolhido.

```
2609 # (se Just pentg) = getControlText (ids!!85)      dial
```

Listagem 5.12 – Leitura do popup que define o pentagrama a editar

Definido, através desta leitura, o pentagrama, pode-se configurar a interface, colocando, por exemplo, se o modo for de apagar, um ícone de borracha à esquerda do pentagrama ativado e um bitmap branco ao lado dos demais. Também se atualiza o registro do pentagrama atual, conforme mostra a listagem 5.13 a seguir:

```
2635 | (pentg=="Pentag 1")
2636 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=0} borracha)) proc
2637 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=190} branco)) proc
2638 # proc = appPIO(appWindowPicture (ids!!93) (drawAt {x=2,y=316} branco)) proc
2639 = {ls = {proc.ls& tipoMouse="borracha.bmp", pentag=pentg}, io = proc.io }
```

Diagrama de anotações para a Listagem 5.13:

- Bitmap branco nos demais pentagramas (aponta para a linha 2637)
- ícone borracha no pentagrama 1 (aponta para a linha 2636)
- Atualização do tipo de edição = "borracha.bmp" (aponta para a linha 2639)
- Atualização do pentagrama, no caso, "Pentag 1" (aponta para a linha 2635)

Listagem 5.13 – Escolha do pentagrama no modo apagar

A figura 5.7 a seguir ilustra a interface após realizar o trecho de código mostrado anteriormente na listagem 5.13.

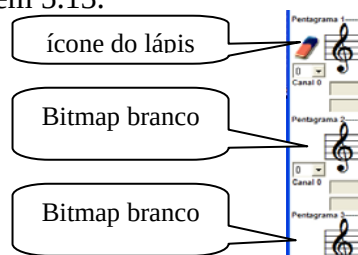


Figura 5.7 – Trecho da interface no modo apagar (borracha)- Pentagrama 1

Configurada a interface para o modo de apagar, o próximo passo é fazer a leitura do ponteiro do mouse para determinar se as coordenadas do mesmo coincidem com as coordenadas de uma nota musical do pentagrama 1 (campo de texto). A criação dos campos das coordenadas x e y do pentagrama 1 e sua localização na interface são mostrados na listagem 5.14 e na figura 5.8

```
EditControl "" (PixelWidth 104) 1 [ControlPos(Above mpx2.zero),ControlId mpx1,ControlSelectState Unable,ControlTip "mpx1"]
+
EditControl "" (PixelWidth 104) 1 [ControlId mpy1,ControlSelectState Unable,ControlTip "mpy1"]
```

Listagem 5.14 – Campos x e y das notas do pentagrama 1

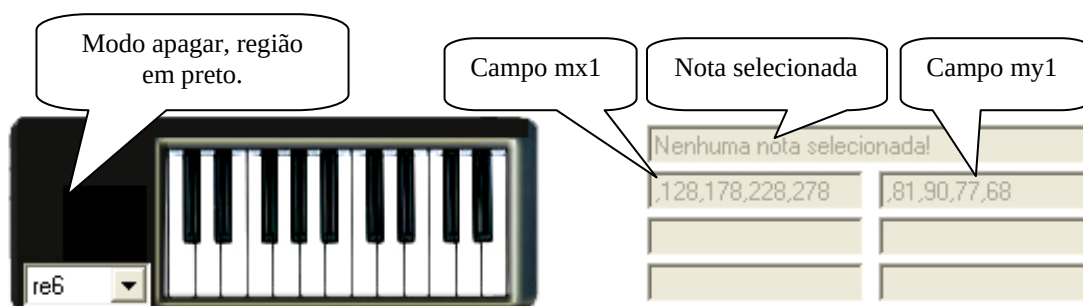


Figura 5.8 – Região de coordenadas do pentagrama 1 – modo apagar

A função que checa se as coordenadas do ponteiro do mouse coincidem com uma coordenada de nota é dada na listagem 5.15 a seguir, onde as devidas explicações deste trecho de programa já constam no código como comentário.

```

2519 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2520 // ESTA FUNÇÃO TEM COMO ARGUMENTO AS COORDENADAS DO MOUSE E DUAS LISTAS
2521 // A PRIMEIRA LISTA É UMA LISTA DE COORDENADAS X DE PONTOS NA JANELA.
2522 // A SEGUNDA DAS COORDENADAS Y DESTE MESMO PONTO
2523 // A FUNÇÃO CHECA SE O PONTEIRO DO MOUSE ESTÁ NA CABEÇA DE UMA NOTA QUE POSSUI
2524 // LARGURA E ALTURA IGUAL A 8 PIXELS. ASSIM, O PONTEIRO DO MOUSE ESTARÁ NA CABEÇA DA NOTA
2525 // SE A COORDENADA x1 e y1 ESTIVEREM EM UMA COORDENADA DAS LISTAS ATÉ 8 PIXELS A MAIS EM X
2526 // E 8 PIXELS A MENOS EM Y (O EIXO Y CRESCE PARA BAIXO NO PENTAGRAMA)
2527 devolveIndiceNota x1 y1 x1 y1
2528 | lista == [] = 0 // se não for uma nota, devolve o inteiro 0
2529 | otherwise = (hd lista)!!2 // se for uma nota, devolve o índice da mesma (o terceiro elemento da lista (i))
2530 where
2531   lista = [[x,y,i] | x<-x1 & y<-y1 & i<-[1..(length x1)] | (x1>=x) &&
2532   (x1<=(x+8)) &&
2533   (y1<= y) &&
2534   (y1>=(y-8))]
2535 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

Listagem 5.15 – Seleção de nota com ponteiro de mouse

5.5 Parâmetros iniciais.

Para montar o arquivo MIDI, necessita-se saber a fórmula de compasso, a armadura de clave, o metrônomo e o valor da ppq. Optou-se por colocar dois valores *pré-setados*, sendo um deles para adequar ao exemplo do Capítulo 2 para validação da interface.

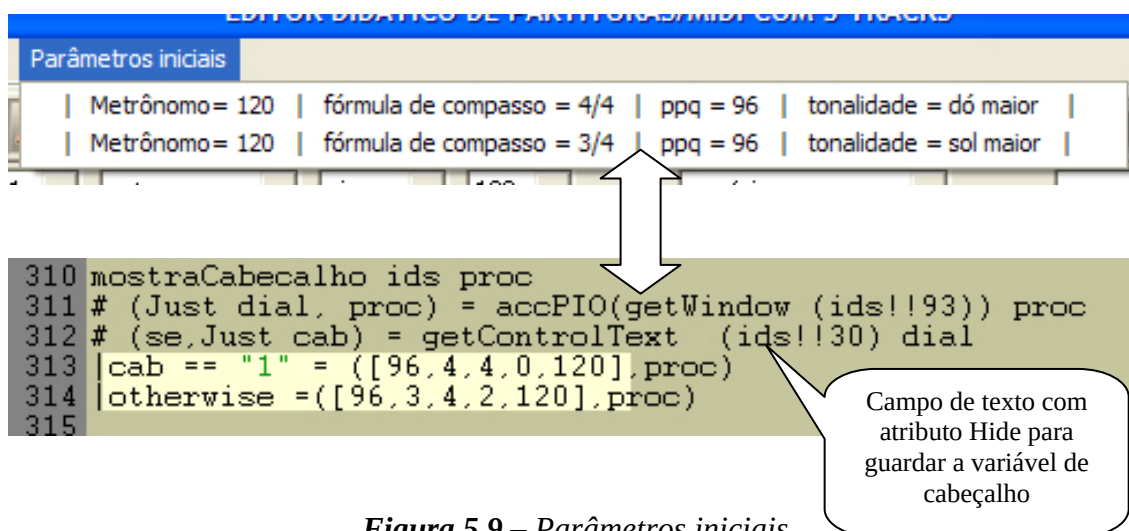


Figura 5.9 – Parâmetros iniciais

5.6 Conclusão.

Procurou-se neste capítulo dar os últimos subsídios básicos para se implementar interfaces visuais utilizando a linguagem Clean para aplicativos multimídia no domínio da tecnologia MIDI e arquivos SMF. No próximo capítulo serão apresentados alguns estudos de casos utilizando os conceitos apresentados neste capítulo e nos anteriores, de tal forma a mostrar as potencialidades das bibliotecas e conceitos já descritos, bem como validá-los.

Capítulo 6

Validação e Estudos de Casos.

Neste capítulo serão realizados estudos de casos empregando as bibliotecas criadas nos capítulos anteriores, bem como os conceitos de interface, gerando novos aplicativos multimídia aplicados ao domínio musical. Assim, será visto:

1. Aplicando a biblioteca ***separaEventosF0*** na identificação e ações em arquivos MIDI formato 0, tais como:
 - Ler as notas de um arquivo SMF formato 0, eliminando o problema de redundância na identificação de notas devido a running status;
 - Dado um arquivo SMF formato 0, converter o código MIDI da notas em nome, e identificar o range do mesmo;
 - Ler, transpor e salvar uma seqüência MIDI de um arquivo SMF formato 0.
2. Aplicando as bibliotecas ***geraMIDIF0*** para validação do exemplo dado no item 2.12 do Capítulo 2.
3. Aplicando as bibliotecas ***geraMIDIF1*** para validação do exemplo dado no item 2.12 do Capítulo 2.
4. Implementação de um Leitor MIDI formato 0 ou 1 e Conversor Didático de arquivos MIDI escritos em Hexadecimal e Decimal para SMF formato 0 ou 1.
5. Implementação de um Editor de Partituras Multi-Instrumental e Multi-Canal. Com salvamento em formato proprietário e em SMF formato 1.

6.1 Aplicando a biblioteca `separaEventosF0` na identificação e ações em arquivos MIDI formato 0.

Esta biblioteca foi criada para que se possa abrir um arquivo MIDI em um formato de listas de eventos, permitindo a análise de seu conteúdo, permitindo edições que venham a alterá-lo conforme desejado, e, posteriormente, salvar o mesmo para que possa ser reproduzido em qualquer equipamento MIDI.

Quando um arquivo MIDI é aberto com esta biblioteca, a mesma cria uma lista de eventos, eliminando o running status, quando houver. Cada evento possui a mesma estrutura:

[**status** , **tipo** , **nBytesOff**, **número de bytes do delta time**, delta-time + evento]

- Na cabeça de cada lista está o código do **status** do evento da mesma, permitindo-se que se consulte de forma simples de qual evento se trata;
- No segundo elemento da lista vem uma característica do **tipo** do evento. Se o evento for de ativar nota, o segundo elemento vem o código da nota; se o status for de um evento de meta-evento, o segundo elemento é o código, o tipo do meta evento, e assim por diante. Ex: [144,**60**,... significa: ativar nota no canal 0(144) e que a nota é a **60** (do5).
- O terceiro elemento **só interessa ao programador da biblioteca**. O mesmo guarda um valor que permite ao programador saber se o evento do arquivo aberto continha ou não running status.
- O quarto elemento indica **quantos bytes de delta time** o evento possui. Ex: [144,60, 4, **2**,... significa, que o evento é de ativar nota no canal 0, nota do5, e que o delta time deste evento possui **2** bytes.
- A partir do quarto byte tem-se a mensagem original, com seu delta time, status e bytes de dados.

Conforme mostrado anteriormente, esta biblioteca permite ao usuário/programador, realizar ações em um arquivo SMF formato 0 e salvá-lo posteriormente. O formato das listas de eventos do arquivo MIDI aberto permite que se possam implementar aplicativos de consulta e transformação (modificação) em arquivos existentes de uma forma simples e aderente utilizando o paradigma funcional, principalmente utilizando notação Zermelo-Fraenkel.

Como estudos de casos desta biblioteca serão apresentados três aplicativos de consulta em SMF e uma implementação de transformação.

6.1.1 – Consultas das notas musicais pertencentes a um SMF formato 0.

Como primeira consulta, será apresentado um aplicativo que devolve uma lista de notas musicais existentes em um determinado arquivo SMF formato 0, para um determinado canal MIDI escolhido. Assim, a partir deste conhecimento, podem-se fazer estudos de harmonia, arranjos, conhecer o range de um determinado instrumento, canal MIDI, bem como outras aplicações mais.

Para fazer este tipo de consulta, deve-se recordar que:

- Ativar nota é um evento que começa com um Byte de status com valores entre 144 (ativar nota no canal 0) e 159 (ativar nota no canal 15).
- Desativar nota é um evento que começa com um Byte de status com valores entre 128 (desativar nota no canal 0) e 143 (desativar nota no canal 15).
- Pode-se utilizar o evento de ativar nota com volume 0 para substituir o evento de desativar nota (interessante quando se utiliza running status), conforme teoria mostrada no capítulo 2.

Para se fazer uma consulta de quais notas existem em um arquivo, é importante recordar como o formato de um evento de ativar nota é devolvido pela biblioteca `separaEventosF0`.

Formato padrão:

[status:dados:NBdrop:nBDT:[deltatime_mensagem]]

Teoricamente, bastaria fazer uma notação Zermelo-Fraenkel que pegasse todas as notas cuja cabeça da lista fosse um status de ativar nota (valores entre 144 e 159). Na prática, deve-se, também, levar em consideração o volume da nota ativada. Isto se dá devido ao fato de que se o volume da nota analisada for igual a zero, isto significa que o evento é de desativar nota (ativar nota com volume zero é equivalente a se desativar esta nota). Assim, evita-se pegar a mesma nota ao ativá-la e desativá-la, o que geraria uma lista de notas com repetições não existentes.

O volume de uma nota musical, em um evento de ativar ou desativar nota, é o último elemento da lista. Assim, para se conhecer o volume, basta conhecer o último elemento da lista⁶⁰.

Observe nos dois exemplos a seguir, como fica o programa de consulta de notas.

Exemplo 1: Programa que não leva em conta se o volume da nota é ou não igual a zero. A listagem 6.1 mostra o programa que abre um SMF (*do_re_mi.mid*, em anexo no CD), o qual não possui evento de ativar nota com volume zero (desativar nota). Desta forma, nenhuma nota é repetida indevidamente.

```
Start proc
#(mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
# todosEventos = listaEventos mensagens
# notasMusicais = [x!!1\\x<-todosEventos | x!!0>=144 && x!!0<=159]
=(notasMusicais,proc)
```

Listagem 6.1 – Função sem tratamento de volume > zero

Ao se executar o programa, tem-se como resultado:

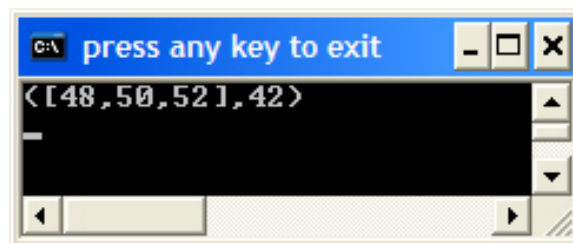


Figura 6.1 – Resultado do arquivo do_re_mi.mid

Onde: 48 é a nota dó4, 50 é nota ré4 e 52 é a nota mi4.

Exemplo 2: Abrindo um SMF, *do_re_miCW.mid*, o qual utiliza o status de ativar nota com volume zero no lugar de ativar nota. Ao rodar o programa, tem-se com o resultado a repetição das notas musicais.

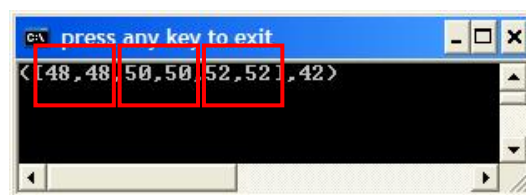


Figura 6.2 – Resultado do arquivo do_re_miCW.mid

⁶⁰ A função *last* de Clean devolve o último elemento de uma lista.

Para evitar a duplicação destas notas na consulta, basta acrescentar na notação Zermello-Fraenkel a restrição no domínio que só se deseja as notas cujo volume for maior que zero.

Modificar o programa, em Clean, utilizando a notação Zermelo-Fraenkel é bastante simples, bastando acrescentar a restrição que o último elemento da lista de ativar notas deve ser maior que zero (*last x > 0*).

```
Start proc
#(mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
# todosEventos = listaEventos mensagens
# notasMusicais = [x!!1\\x<-todosEventos | x!!0>=144 && x!!0<=159 && last x >0]
=(notasMusicais,proc)
```

Listagem 6.2 – Programa modificado

Ao rodar o programa, com tal restrição, para o mesmo arquivo anterior, tem-se como resposta uma lista de notas sem repetição (conforme arquivo SMF aberto):

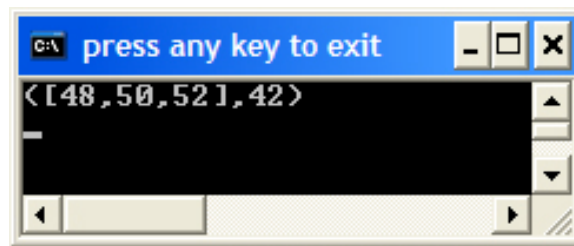


Figura 6.3 – Resultado corrigido do arquivo do_re_miCW.mid

6.1.2 - Devolvendo nomes das notas musicais ao invés dos códigos MIDI.

Podem-se conhecer os nomes das notas musicais do arquivo aberto em vez de seus respectivos códigos MIDI. Para tanto, basta acrescentar na notação Zermello-Fraenkel, na regra da função, uma conversão de código MIDI para nome de nota. Antes a função tinha como regra apenas devolver o segundo elemento da lista (**x!!1**), o qual possuía o código da nota musical. A regra, agora, é devolver o nome da nota, cujo índice é o código MIDI da nota (**notas!!(x!!1)**). Assim, implementou-se na biblioteca **separaEventosF0**, uma lista de nome de notas musicais, de tal forma que a cada código MIDI corresponde um determinado nome de nota.

```

652 notas =
653 ["do0", "do#0", "re0", "re#0", "mi0", "fa0", "fa#0", "sol0", "sol#0", "la0", "la#0", "si0",
654 "do1", "do#1", "re1", "re#1", "mi1", "fa1", "fa#1", "sol1", "sol#1", "la1", "la#1", "si1",
655 "do2", "do#2", "re2", "re#2", "mi2", "fa2", "fa#2", "sol2", "sol#2", "la2", "la#2", "si2",
656 "do3", "do#3", "re3", "re#3", "mi3", "fa3", "fa#3", "sol3", "sol#3", "la3", "la#3", "si3",
657 "do4", "do#4", "re4", "re#4", "mi4", "fa4", "fa#4", "sol4", "sol#4", "la4", "la#4", "si4",
658 "do5", "do#5", "re5", "re#5", "mi5", "fa5", "fa#5", "sol5", "sol#5", "la5", "la#5", "si5",
659 "do6", "do#6", "re6", "re#6", "mi6", "fa6", "fa#6", "sol6", "sol#6", "la6", "la#6", "si6",
660 "do7", "do#7", "re7", "re#7", "mi0", "fa7", "fa#7", "sol7", "sol#7", "la7", "la#7", "si7",
661 "do8", "do#8", "re8", "re#8", "mi8", "fa8", "fa#8", "sol8", "sol#8", "la8", "la#8", "si8",
662 "do9", "do#9", "re9", "re#9", "mi9", "fa9", "fa#9", "sol9", "sol#9", "la9", "la#9", "si9",
663 "do10", "do#10", "re10", "re#10", "mi10", "fa10", "fa#10", "sol10", "sol#10", "la10", "la#10", "si10"]

```

Listagem 6.3 – Lista com nome das notas

De posse do código da nota musical, basta acessar na lista de nomes de notas a nota cujo índice foi localizado na sequência musical lida.

Assim, **notas!!60 = “do5”**, **notas!!64 = “mi5”**, e assim por diante. Desta forma, a regra da função é pegar, de uma lista de notas, uma nota musical cujo índice é o código em inteiro da nota musical representada pelo segundo elemento da lista do evento de ativar nota, ou seja: **notas!!(x!!1)**.

O programa de exibir os nomes das notas existentes em um arquivo SMF formato 0 é dado na listagem 6.4 a seguir:

```

5 Start proc
6 #(mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
7 # todosEventos = listaEventos mensagens
8 # notasMusicais = [notas!!(x!!1)]\\x<-todosEventos | x!!0>=144 && x!!0<=159 && last x >0]
9 =(notasMusicais,proc)

```

Listagem 6.4 – Programa – Exibe o nome às notas

Executando o programa, têm-se nomes de notas em vez de seus códigos, conforme mostrado a seguir:

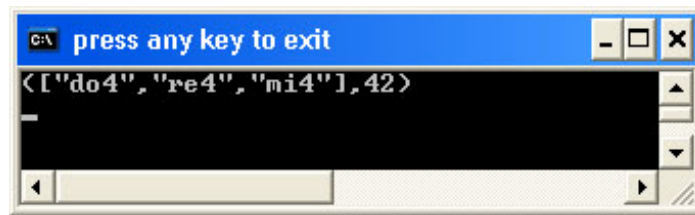


Figura 6.4 – Resultado – nome das notas

Observe na linha 8 do programa, a aderência da linguagem a este tipo de consulta e na transformação nos dados (códigos em nomes de notas), permitindo extrema flexibilização nas modificações e acréscimos a um programa já existente. Esta assertiva pode ser observada no próximo exemplo de consulta: Determinando o range de um canal MIDI especificado.

6.1.3 Determinando o range de um canal MIDI especificado.

Como segunda consulta, partindo dos conceitos adquiridos na primeira: determinar as notas musicais de um SMF, um aplicativo útil é conhecer o range musical de cada canal, cada instrumento MIDI, para saber se um determinado instrumento acústico, ou um cantor, poderá ou não tocar a melodia na tonalidade em que está. Assim, a seguir é mostrado um programa de como determinar o range musical de cada canal utilizando esta biblioteca em estudo.

A figura 6.5 mostra um pentagrama de uma música escrita no formato MIDI utilizando o canal zero. Observe que o range das notas musicais vai da nota dó5 até a nota si6.



Figura 6.5 – Pentagrama

Utilizando a lista de notas musicais colocada na biblioteca, mostrada anteriormente, e os conceitos do exemplo anterior, implementou-se o programa que mostra o range das notas de um determinado canal MIDI. Para tanto, basta apenas ordenar as notas musicais, de acordo com seu código MIDI⁶¹, e pegar o primeiro e o último elemento da lista ordenada⁶². Neste caso, não importa o tipo de mensagem de desativar nota, já que, mesmo se forem pegadas notas repetidas, o que interessa é a nota de menor código (limite inferior do range) e a nota de maior código (limite superior do range).

O programa que faz a consulta do range é mostrado a seguir:

```

4 Start proc
5 # canal = 0
6 # (mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
7 # todosEventos = listaEventos mensagens
8 # notasMusicais = [(x!!1)\x<-todosEventos | (x!!0)==(144+canal)]
9 # notasOrdenadas = sort(notasMusicais)
10 # range = [notas!!(hd notasOrdenadas),notas!!(last notasOrdenadas)]
11 =(range,proc)

```

Listagem 6.5 – Consulta do range

⁶¹ A função em Clean que ordena uma lista é **sort**

⁶² A função que pega o primeiro elemento de uma lista é **hd**, e a função que pega o último elemento de uma lista, em Clean, é **last**.

Ao executá-lo no arquivo SMF range.mid, o resultado é o seguinte:

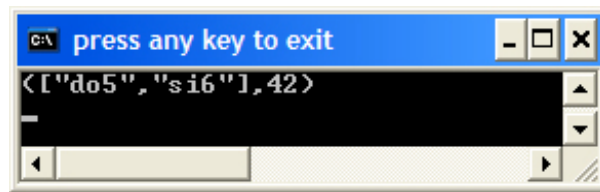


Figura 6.6 – Resultado – consulta do range

O qual confirma o range mostrado na figura 6.5.

6.1.4 Consulta sobre as mensagens de meta eventos.

Como terceira consulta, é implementada uma função que devolve todos os meta eventos de um arquivo SMF formato 0. De posse desta informação, pode-se, por exemplo, extrair o lirismo do arquivo MIDI, caso o mesmo possua, conhecer a fórmula de compasso, analisar a dinâmica de tempo ou volume, conhecer a tonalidade, e outras características do arquivo. Para tanto, em vez de apenas escolher os eventos que tenham o código 255 na cabeça da lista, poder-se-ia analisar também o segundo elemento e filtrar apenas o meta evento desejado. Nesta consulta, em vez de mostrar a lista formatada pela biblioteca, serão mostradas listas contendo os meta eventos na forma com que foram criados, ou seja, eliminando os 4 (quatro) primeiros Bytes da lista de eventos. Assim, a regra da função da notação Zermello-Fraenkel desta consulta é devolver a lista sem os primeiros 4 Bytes da mesma, como segue.

```
Start proc
#(mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
# todosEventos = listaEventos mensagens
# metaEventos = [ drop 4 x \\ x<-todosEventos | x!!0==255
=(metaEventos,proc)
```

Listagem 6.6 – Meta eventos

Aplicando este programa no arquivo MIDI *geraMidiF0.mid*, pode-se observar que o mesmo devolve exatamente os meta eventos lá registrados.

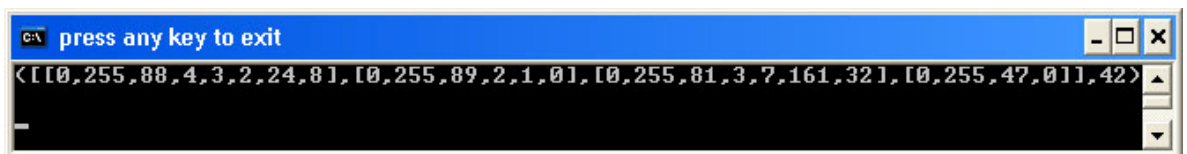


Figura 6.7 – Resultado do programa “metaEventos”

6.1.5 Transformação – Transpondo uma música de um SMF.

Dentre várias aplicações, utilizar-se-á como exemplo, editar, modificar um arquivo SMF formato 0 já existente. A modificação escolhida é a ação de transpor uma seqüência musical, um arquivo MIDI, em semitons, conforme especificado pelo usuário. Para tanto, deve-se ter em mente os seguintes tópicos:

1. Não se pode transpor o décimo canal MIDI (canal 9), já que o mesmo sempre será o canal da bateria. Neste canal, cada nota musical equivale a um instrumento de percussão, e, desta forma, se o canal sofrer transposição, os instrumentos serão modificados. Como exemplo, se o canal 9 for transposto em dois semitons, o bumbo vira caixa e a bateria ficará sem a marcação forte do mesmo.
2. No protocolo MIDI, para cada evento de ativar nota tem-se um evento de desativar a mesma nota. Assim, ao transpor uma nota musical, deve-se transpor a nota no evento de ativar e no evento de desativar nota, possuindo running status ou não, estes eventos.
3. No padrão MIDI existem somente 128 notas musicais (código 0 a 127). Assim, deve-se observar que, ao transpor uma nota musical, o código da nota transposta não ultrapasse o valor 127, e, no caso de transposição negativa, que a nota não fique com o código menor que zero. Assim, caso isto ocorra, deve-se escolher uma ação, como, por exemplo:
 - Não realizar a transposição e enviar uma mensagem que a transposição não é possível por a seqüência musical sair do range MIDI;
 - Realizar a transposição oitavando (acima ou abaixo) a nota transposta;
 - Outra opção harmonicamente válida, predefinida ou à escolha do usuário.

É importante relembrar a estrutura da lista que contém uma nota musical:

- Ativar nota

[status_de_ativar_nota, **nota**, NBdrop, NBDT, delta time, status_de_ativar_nota, **nota**, volume]

- Desativar nota

[status_de_desativar_nota, **nota**, NBdrop, NBDT, delta time, status_de_desativar_nota, **nota**, volume]

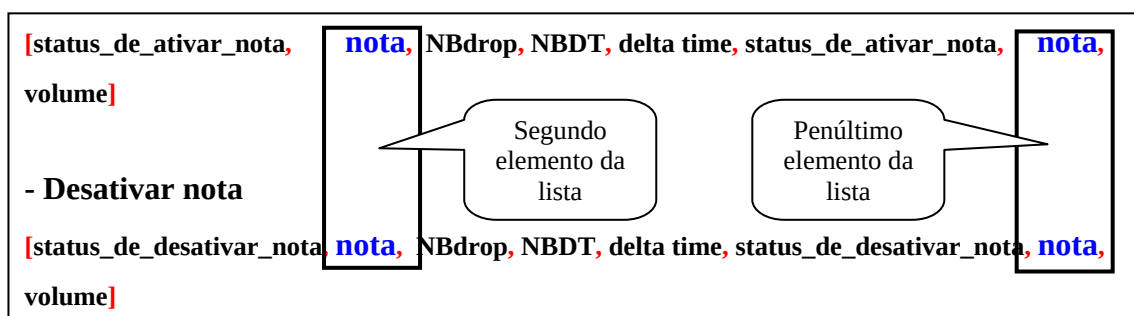


Figura 6.8 – Evento de Ativar e Desativar notas

Observe que, tanto no ato de ativar quanto desativar nota, devem-se alterar dois elementos da lista, os quais contêm a informação da nota musical. Os elementos são o segundo e o penúltimo de cada uma destas listas.

Assim, para modificar, transpor estes elementos, basta substituí-los pela nova nota, onde:

$\text{novaNota} = \text{código_da_nota_original} + \text{número_de_semitons.}$

A função que troca um elemento de uma lista é:

updateAt

O formato desta função é:

updateAt índice_do_elemento novo_elemento lista

No caso, portanto, o índice da primeira ocorrência de nota do evento a ser trocado é 1, ou seja, o segundo elemento da lista.

Para mudar a segunda ocorrência de nota neste evento, o índice dele é o do penúltimo elemento, ou seja, o índice 1 do reverso da lista.

Assim, para atualizar, modificar a primeira ocorrência, basta utilizar a função updateAt da seguinte forma:

updateAt 1 novo_elemento lista

Para atualizar a segunda ocorrência, a função updateAt fica:

reverse (updateAt í novo_elemento (reverse lista))

Ou seja, inverte-se a lista (*reverse lista*) e atualiza-se seu segundo elemento (que era o penúltimo da lista normal). Feito isto, tem-se a lista atualizada, mas invertida. Assim, reverte-se novamente esta lista para ficar no formato original.

Exemplo:

Dada a lista de ativar a nota dó5 (código 60), com volume 100 e com delta time de 1 Byte (valor 120).

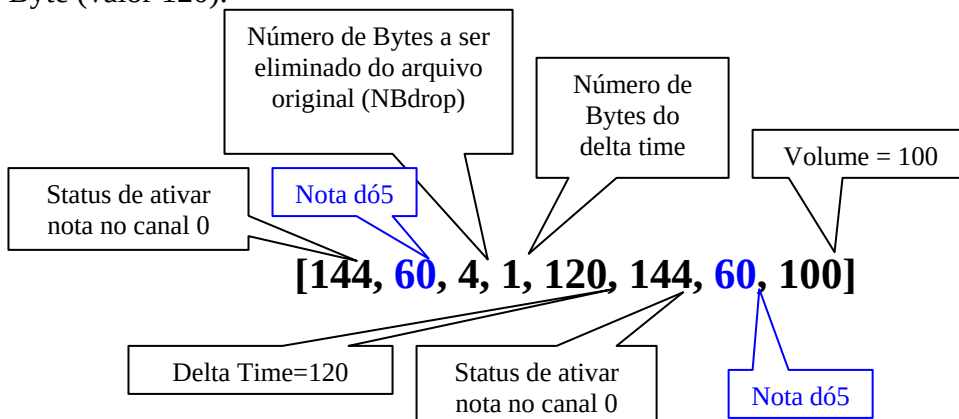


Figura 6.9 – Lista de ativar nota

Observe que, neste caso, deve-se modificar a nota 60(dó5) para transpô-la. Adote transpor esta nota em 2 semitons, ou seja, o novo código deverá ser o código original, 60, acrescido de duas unidades, resultando no código 62 (ré5). Assim:

1. Modifique a primeira ocorrência da nota:

updateAt 1 (60+2) [144, 60, 4, 1, 120, 144, 60, 100] = [144, 62, 4, 1, 120, 144, 60, 100]

2- Modifique a segunda ocorrência da nota. Primeiro inverta a lista original utilizando a função do Clean reverse, ou seja:

reverse [144, 62, 4, 1, 120, 144, 60, 100] = [100, 60, 144, 120, 1, 4, 62, 144]

3 - Feito isto, modifique o segundo elemento (índice 1) desta lista:

updateAt 1 (60+2) [100, 60, 144, 120, 1, 4, 62, 144] = [100, 60, 144, 120, 1, 4, 62, 144]

4 - Observe que a lista ficou invertida, devendo novamente fazer sua reversão:

reverse [100, 60, 144, 120, 1, 4, 62, 144] = [144, 62, 4, 1, 120, 144, 62, 100]

Assim, conclui-se o processo de transposição de uma nota musical em uma lista de eventos, sem alterar o formato de abertura do arquivo feito pela biblioteca. Pode-se fazer isto de várias formas, a maioria aderente ao pensamento e à forma com que um músico realiza tal tarefa.

6.1.6 - Implementando o aplicativo que faz a transposição de uma música em semitons, conforme explicado anteriormente:

Neste aplicativo, as notas musicais serão modificadas, transpostas, conforme número de semitons fornecido. Assim, para a comprovação, validação, do que se pretende fazer, devolver-se-á uma tupla com quatro elementos:

- 1- Uma lista com os códigos das notas originais;
- 2- Uma lista com os códigos das notas transpostas;
- 3- Uma lista dos eventos de notas originais;
- 4- Uma lista dos eventos de notas já transposta.

Para mostrar simplicidade do programa, não será inicialmente tratado o problema do canal 9 nem a checagem se o código da nota ultrapassa o valor 127 ou fica abaixo de 0. Posteriormente, logo após esta implementação, será mostrado o programa com o tratamento destas exceções.

O ARQUIVO MIDI UTILIZADO

- A figura 6.10 mostra a partitura utilizada como exemplo deste caso:



Figura 6.10 – Partitura exemplo

- O código em decimal dos eventos desta partitura, de seu arquivo SMF é:

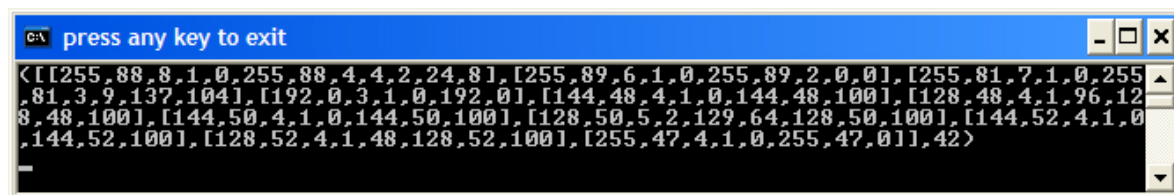


Figura 6.11 – Código em decimal dos eventos da partitura

- O programa para transpor a música em semitons, deve fazer o seguinte:

1. Escolher o número de semitons para transpor:

semitons = 2

2. Abrir e criar uma lista de códigos do arquivo MIDI escolhido (formato 0):

(mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc

3. Pegar a lista e separar seus eventos, cada um em uma lista, conforme formato já descrito:

musicaOriginal = listaEventos mensagens

Onde a função **listaEventos** é uma função da biblioteca deste estudo de caso.

4. Transpor as notas musicais dos eventos de ativar e desativar nota.

musicaTransposta = [transpoeEventoNota listaEventos semitons \\ listaEventos <- musicaOriginal]

O programa completo fica:

```

5 Start proc
6 # semitons = 2
7 # (mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
8 # musicaOriginal = listaEventos mensagens
9 # musicaTransposta = [transpoeEventoNota listaEventos semitons \\ listaEventos <- musicaOriginal
10 # notasOriginais = [x!!1\\x<-musicaOriginal | x!!0>=144 && x!!0<=159 && last x >0]
11 # notasTranspostas = [x!!1\\x<-musicaTransposta | x!!0>=144 && x!!0<=159 && last x >0]
12 # eventosNotasOriginais = [x \\x<-musicaOriginal | x!!0>=144 && x!!0<=159 && last x >0]
13 # eventosNotasTranspostas = [x \\x<-musicaTransposta | x!!0>=144 && x!!0<=159 && last x >0]
14 =(stLinha,notasOriginais,stLinha,notasTranspostas,stLinha,eventosNotasOriginais,
15 |stLinha,eventosNotasTranspostas,stLinha,proc)
16
17 transpoeEventoNota :: [Int] Int -> [Int]
18 transpoeEventoNota listaEventos semitons
19 |(listaEventos!!0>=128 && listaEventos!!0<=159)= transpoe listaEventos semitons
20 |otherwise = listaEventos
21
22 transpoe :: [Int] Int -> [Int]
23 transpoe listaEventos semitons
24 #update1 = updateAt 1 (notaOriginal+semitons) listaEventos
25 #update2 = reverse (updateAt 1 (notaOriginal+semitons) (reverse update1) )
26 =update2
27 where
28 notaOriginal = listaEventos!!1
29 novaNota = (notaOriginal + semitons)
30
31 stLinha :: {Char}
32 stLinha = {toChar 10,toChar 13}

```

Listagem 6.7 – Programa completo de transposição

Observe, novamente, que, para mostrar as notas, colocou-se a restrição que o volume tem que ser maior que 0 (last x >0), já que se podem ter eventos de desativar nota utilizando o evento de ativar nota com volume 0. Se não for colocada esta restrição, algumas notas apareceriam duplicadas, conforme já explicado anteriormente.

Ao rodar o programa, obtém-se o seguinte resultado para a música escolhida (do_re_miCW.mid).

pr any key to exit

```
[48,50,52],<'
[50,52,54],<'
[[144,48,4,1,0,144,48,100],[144,50,3,1,0,144,50,100],[144,52,3,1,0,144,52,100]
<'
[[144,50,4,1,0,144,50,100],[144,52,3,1,0,144,52,100],[144,54,3,1,0,144,54,100]
<'
42>
```

```
7 # (mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
8 # musicaOriginal = listaEventos mensagens
9 # musicaTransposta = transpoeEventoNota listaEventos semitons \\ listaEventos <- musicaOriginal ]
10 # notasOriginais = [x!!0>=144 && x!!0<=159 && last x >0]
11 # notasTranspostas = [x!!0>=144 && x!!0<=159 && last x >0]
12 # eventosNotasOriginais = [x!!0>=144 && x!!0<=159 && last x >0]
13 # eventosNotasTranspostas = [x!!0>=144 && x!!0<=159 && last x >0]
14 = (stLinha,notasOriginais,notasTranspostas, eventosNotasOriginais, eventosNotasTranspostas, stLinha, eventosNotasOriginais, eventosNotasTranspostas)
15
16
17 transpoeEventoNota :: [Int] Int -> [Int]
18 transpoeEventoNota listaEventos semitons
19 | (listaEventos!!0)>=128 && listaEventos!!0<=159 = transpoe listaEventos semitons
20 | otherwise = listaEventos
21
22 transpoe :: [Int] Int -> [Int]
23 transpoe listaEventos semitons
24 | (novaNota > 127)
25   #update1 = updateAt 1 (notaOriginal+semitons-12) listaEventos
26   #update2 = reverse (updateAt 1 (notaOriginal+semitons-12) (reverse update1) )
27   =update2
28 | (novaNota < 0)
29   #update1 = updateAt 1 (notaOriginal+semitons+12) listaEventos
30   #update2 = reverse (updateAt 1 (notaOriginal+semitons+12) (reverse update1) )
31   =update2
32 | otherwise
33   #update1 = updateAt 1 (notaOriginal+semitons) listaEventos
34   #update2 = reverse (updateAt 1 (notaOriginal+semitons) (reverse update1) )
35   =update2
36 where
37   notaOriginal = listaEventos!!1
38   novaNota     = (notaOriginal + semitons)
39
40 stLinha :: {Char}
41 stLinha = {toChar 10,toChar 13}
```

Listagem 6.8 – Programa, e resultado da transposição

Observações:

- 1 Quando o código da nota transposta for >127, transpõe-se a mesma, uma oitava abaixo (subtrai-se 12 de seu código).
- 2 Quando o código da nota transposta for < 0, transpõe-se a mesma, uma oitava acima (adiciona-se 12 ao seu código).

Um programa para abrir o arquivo MIDI, transpô-lo, tocar e salvar o resultado da transposição é mostrado logo a seguir, onde a função de transposição mostrada anteriormente foi acrescida à biblioteca `separaEventosF0`.

```
5 Start proc
6 # semitons = 2
7 # (mensagens,cabecalho,endMIDI,proc) = abrirEventosMIDI proc
8 # musicaOriginal = listaEventos mensagens
9 # musicaTransposta = [transpoeEventoNota listaEventos semitons \\ listaEventos <- musicaOriginal ]
10 # proc = salvarArquivoMIDI cabecalho musicaTransposta proc
11 # (_, proc) = OSStopMusic proc
12 # (_, proc) = playMIDISalvo proc
13 =proc
```

Listagem 6.9 – Programa que abre, transpõe, toca e salva o arquivo MIDI

Pode-se perceber, assim, que manipular a estrutura de um arquivo aberto é bastante simples e aderente ao paradigma funcional e a linguagem Clean adotada. Com poucas linhas de código podem-se fazer consultas e transformações complexas, gerando um código legível e fácil de ser analisado e modificado.

6.2 Aplicando as bibliotecas geraMidiF0 para validação do exemplo dado no item 2.12 do Capítulo 2.

Para comprovar o desempenho das funções implementadas para geração de arquivos formato 0, será utilizado o exemplo do capítulo 2 desta dissertação, item 2.12, o qual apresenta uma grade orquestral com três instrumentos, onde um instrumento, o violão, está registrado em um canal MIDI, e, os outros dois, flautas, estão referenciados no mesmo canal MIDI⁶³. Neste exemplo têm-se vários conceitos a serem implementados e provados, tais como:

- Pausas no início, no meio e no fim de compassos;
- Acordes;
- Figuras musicais de tempo diferentes, gerando delta times com 1 e mais bytes;
- Fórmula de compasso e tonalidade diferentes da padrão;
- Dois tracks utilizando o mesmo canal MIDI;

A figura 6.12 mostra novamente a grade orquestral em questão



Figura 6.12 – Grade Orquestral

6.2.1 - Arquivo MIDI Obtido Teoricamente, Conforme Capítulo 2, Item 2.12.

Utilizando os valores dos parâmetros iniciais:

- tonalidade de sol maior,
- ppq = 96
- metrônomo = 120 (setTime = 07 161 32),
- fórmula de compasso $\frac{3}{4}$,
- Instrumento do canal 0 = violão (24)
- Instrumento do canal 1 = flauta (73).

⁶³ Poder-se-ia ter escolhido dois canais diferentes, apesar de serem instrumentos iguais.

O código obtido teoricamente, no capítulo 2, item 2.12, com tipo de dados Inteiro, para o arquivo MIDI SMF formato 0 desta grade orquestral, é:

```
77 84 104 100 0 0 0 6 0 0 0 1 0 96 77 84 114 107 0 0 0 79 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88
4 3 2 24 8 0 192 24 0 193 73 0 145 62 100 0 144 69 100 96 128 69 0 0 145 67 100 48 129 67 0 0 145
64 100 48 129 64 0 0 129 62 0 0 144 69 100 0 144 72 100 96 128 69 0 0 128 72 0 0 255 47 0
```

Figura 6.13 – Código em Decimal utilizando a função geraMidiF0

6.2.2 Arquivo MIDI obtido pela função geraMidiF0 implementada nesta dissertação.

A função **geraMidiF0** possui sete argumentos e devolve um vetor de caracteres (o arquivo MIDI), conforme discriminado a seguir:

```
geraMidiF0 :: [Int] [Int] [{#Char}] [Int] [{#Char}] [Int] [Int] -> {#Char}
geraMidiF0 cabeçalho ativaDesativa programChange deltaTime nota volume canal
```

onde:

FUNÇÃO: **geraMidiF0**

ARGUMENTOS DA FUNÇÃO **geraMidiF0**

- **cabeçalho**: uma lista de inteiros, contendo:
 - a ppq,
 - o numerador da fórmula de compasso,
 - o denominador da fórmula de compasso,
 - armadura de clave (tonalidade - 30 tipos) (ver Anexo III)
 - o metrônomo
 Exemplo: **cabeçalho** = [96, 3, 4, 2, 120]
- **ativaDesativa**: uma lista de ativa (1) e desativa (0) notas
Exemplo: **ativaDesativa** = [1,1,0,1]
- **programChange** : uma lista com nome de instrumentos por nota
Exemplo: **programChange** = ["flauta","violao","violao","flauta"]
- **deltaTime**: uma lista com valores inteiros de delta times
Exemplo: **deltaTime** = [0,0,96,0]
- **nota**: uma lista de notas musicais
Exemplo: **nota** = ["re5","la5","la5","sol5"]
- **volume**: uma lista de valores inteiros de volume de 0 a 127
Exemplo: **volume** = [100,100,0,100]
- **canal**: uma lista de canais (de 0 a 15)
Exemplo: **canal** = [1, 0, 0,1]

Assim, a estrutura dos dados para gerar o arquivo SMF desejado, conforme exemplo do capítulo 2, é o que segue:

```
# cabecalho = [
    96, // ppq
    3, // numerador da formula de compasso
    4, // denominador da formula de compasso
    2, // armadura de clave
    120 // metronomo
]

# ativaDesativa = [ 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0 ]
# programChange = ["flauta", "violao", "violao", "flauta", "flauta", "flauta", "flauta", "flauta", "violao", "violao", "violao", "violao"]
# deltaTime = [ 0, 0, 96, 0, 48, 0, 48, 0, 0, 0, 96, 0 ]
# nota = ["re5", "la5", "la5", "sol5", "sol5", "mi5", "mi5", "re5", "la5", "do6", "la5", "do6"]
# volume = [ 100, 100, 0, 100, 0, 100, 0, 0, 100, 100, 0, 0 ]
# canal = [ 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0 ]
```

Listagem 6.10 – Estrutura dos dados para gerar o arquivo MIDI

Ao rodar um programa que aplica a função **geraMidiF0** nestas listas de dados, obtém-se o mesmo resultado obtido teoricamente, como se pode ver logo a seguir:

RESULTADO TEÓRICO

```
77 84 104 100 0 0 0 6 0 0 0 1 0 96 77 84 114 107 0 0 0 79 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88
4 3 2 24 8 0 193 73 0 145 62 100 0 144 69 100 96 128 69 0 0 145 67 100 48 129 67 0 0 145
64 100 48 129 64 0 0 129 62 0 0 144 69 100 0 144 72 100 96 128 69 0 0 128 72 0 0 255 47 0
```

Figura 6.14 – Resultado obtido teoricamente

RESULTADO OBTIDO COM A FUNÇÃO **geraMidiF0**

(é mostrado como [Int] em vez de {Char} para que se possa comparar com o resultado teórico)

```
[77,84,104,100,0,0,0,6,0,0,0,1,0,96,77,84,114,107,0,0,0,85,0,255,88,4,3,2,24,8,0,255,89,2,1,0,0,255,81,
3,7,161,32,0,193,73,0,145,62,100,0,192,24,0,144,69,100,96,128,69,0,0,193,75,0,145,67,100,48,129,67,
0,0,145,64,100,48,129,64,0,0,129,62,0,0,192,24,0,144,69,100,0,144,72,100,96,128,69,0,0,128,72,0,0,2
55,47,0]
```

Figura 6.15 – Resultado obtido com a função **geraMidiF0**

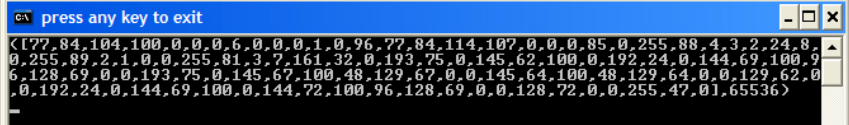
A única diferença entre os dois resultados é que, na função **geraMidiF0**, a cada nota disparada pode-se escolher um instrumento diferente. Não foi tratado o caso de que, caso o instrumento da nota anterior for o mesmo da nota atual, não se colocar mudança de instrumento novamente, o que se pode, posteriormente, implementar com relativa facilidade. Este caso só foi detectado neste momento da validação. Na função de gerar formato 1 isto já foi implementado.

A listagem 6.11 mostra o programa utilizado para rodar a função **geraMidiF0**, com o respectivo resultado obtido da aplicação da mesma nas listas de dados anteriormente descritas:

```

Start proc
# cabecalho = [
    96, // ppq
    3, // numerador da formula de compasso
    4, // denominador da formula de compasso
    2, // armadura de clave
    120 // metronomo
]
# ativaDesativa = [ 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0 ]
# programChange = ["flauta", "violao", "violao", "flauta", "flauta", "flauta", "flauta", "violao", "violao", "violao", "violao" ]
# deltaTime = [ 0, 0, 96, 0, 48, 0, 48, 0, 0, 0, 96, 0 ]
# nota = ["re5", "la5", "la5", "sol5", "sol5", "mi5", "mi5", "re5", "la5", "do6", "la5", "do6" ]
# volume = [ 100, 100, 0, 100, 0, 100, 0, 0, 100, 100, 0, 0 ]
# canal = [ 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0 ]
# listaMIDI = geraMidiF0 cabecalho ativaDesativa programChange deltaTime nota volume canal
# arqSalvar = map toChar listaMIDI
# vetorSalva = {x\\x<-arqSalvar}
# (_,file3,proc)= fopen "C:\\geraMidiF0.mid" FWriteData proc
# file3 = fwrites vetorSalva file3
# (_,proc)= fclose file3 proc
# (_,proc)= OSStopMusic proc
# (_,proc)= playMid "C:\\geraMidiF0.mid" proc
= (listaMIDI,proc)

```



Listagem 6.11 – Programa utilizado para rodar a função *geraMidiF0*

6.3 Aplicando as bibliotecas geraMidiF1 para validação do exemplo dado no item 2.12 do Capítulo 2.

6.3.1 Arquivo MIDI Obtido Teoricamente, Conforme Capítulo 2, Item 2.12.

Utilizando os mesmos valores dos parâmetros iniciais:

- tonalidade de sol maior,
- ppq = 96
- metrônomo = 120 (setTime = 07 161 32),
- fórmula de compasso $\frac{3}{4}$,
- Instrumento do canal 0 = violão (24)
- Instrumento do canal 1 = flauta (73).

O código obtido teoricamente, no capítulo 2, item 2.12, com tipo de dados Inteiro, para o arquivo MIDI SMF formato 1 desta grade orquestral, é:

```
77 84 104 100 0 0 0 6 0 1 0 4 0 96 77 84 114 107 0 0 0 25 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88
4 3 2 24 8 0 255 47 0 77 84 114 107 0 0 0 31 0 192 24 0 144 69 100 96 128 69 0 96 144 69 100 0 144
72 100 96 128 69 0 0 128 72 0 0 255 47 0 77 84 114 107 0 0 0 23 0 193 73 96 145 67 100 48 129 67 0
0 145 64 100 48 129 64 0 0 255 47 0 77 84 114 107 0 0 0 16 0 193 73 0 145 62 100 129 64 129 62 0 0
255 47 0
```

Figura 6.16 – Resultado Teórico

6.3.2 Arquivo MIDI Obtido pela Função geraMidiF1 Implementada nesta Dissertação.

A função **geraMidiF1** possui apenas um argumento: uma lista de listas de listas de tipos Inteiros e devolve um vetor de caracteres (o arquivo MIDI).

```
geraMidiF1 :: [[[Int]]]-> {Char}
geraMidiF1 musica
```

O argumento é uma lista que contém:

- O primeiro elemento: o cabeçalho ([Int]), o qual contém os 5 parâmetros iniciais, a saber:

cabeçalho = [ppq,

**numerador da fórmula de compasso,
denominador da fórmula de compasso,
armadura de clave (tonalidade -Anexo III),
metrônomo]**

Exemplo: **cabeçalho** = [96, 3, 4, 2,120]

- Várias listas com os parâmetros de cada track (listas de inteiros), onde:

Estrutura doTrack = [**Lista de status nota/acorde,**
Lista de instrumentos,
Lista de figuras musicais,
Lista de notas musicais,
Lista de volumes,
Lista de canais]]

Exemplo:

Track inicial =

[[nota,nota, nota,nota, nota,nota,acorde,acorde],
[flauta, flauta, flauta, flauta, flauta, flauta, flauta, flauta],
[semínima,semínima, semínima,semínima,mínima,mínima,semibreve,semibreve],
[do5,re5,la5,sol5,mi5,re#5,pausa,re#5, la5],
[100,120,127,125,125,127,127,127],
[0,0,1,0,1]]

Track convertido para lista de inteiros

[[1,1,1,1,1,1,2,2],
[78,78,78,78,78,78,78,78],
[4,4,4,4,2,2,1,1],
[60,62,67,64,63,500,63,67],
[100,120,127,125,125,127,127,127],
[0,0,1,0,1]]

Tanto o cabeçalho quanto os tracks farão parte da lista do argumento de **geraMidiF1**, e, desta forma, devem ter o mesmo tipo de dado. Assim, como o track é do tipo **[[Int]]**, o cabeçalho (**[Int]**) também deveria ser, devendo, portanto, o mesmo ser colocado dentro de outra lista.

Exemplo: **cabeçalho** = **[[96, 3, 4, 2,120]]**

Para se converter figuras e notas musicais (dadas em string), devem-se convertê-las para seus códigos equivalentes em inteiros. Isto pode ser feito pelas seguintes funções:

- Pega código do instrumento: *pegaCodigoInstr*

Como exemplo, utilizou-se apenas 6 instrumentos, mas pode-se utilizar uma função que pegue o índice de uma lista com todos os instrumentos no padrão GM (General MIDI).

```

205 pegaCodigoInstr :: {#.Char} -> .Int
206 pegaCodigoInstr x =: "Piano" = 0
207 pegaCodigoInstr x =: "Flauta" = 75
208 pegaCodigoInstr x =: "Violino" = 40
209 pegaCodigoInstr x =: "Saxofone" = 64
210 pegaCodigoInstr x =: "Violão" = 24
211 pegaCodigoInstr x =: "Trompete" = 56
212

```

Listagem 6.12 – Função pegaCodigoInstr

- Pega código das notas: notasCod

Neste caso, esta função pega o índice de uma das 128 notas do padrão GM. O índice possui o mesmo valor do código do instrumento.

```

213 notasCod :: [Int]
214 notasCod = indexList notaSB
215
216 pegaCodigoNotaInt :: {#.Char} -> Int
217 pegaCodigoNotaInt nt = hd [snd(x,y) \\ x<- notaSB & y<- notasCod | x == nt]
218
219 notaSB :: [.{#.Char}]
220 notaSB =
221 ["do0", "re0", "re#0", "mi0", "fa0", "fa#0", "sol0", "sol#0", "la0", "la#0", "si0",
222  "do1", "do#1", "re1", "re#1", "mi1", "fa1", "fa#1", "sol1", "sol#1", "la1", "la#1", "si1",
223  "do2", "do#2", "re2", "re#2", "mi2", "fa2", "fa#2", "sol2", "sol#2", "la2", "la#2", "si2",
224  "do3", "do#3", "re3", "re#3", "mi3", "fa3", "fa#3", "sol3", "sol#3", "la3", "la#3", "si3",
225  "do4", "do#4", "re4", "re#4", "mi4", "fa4", "fa#4", "sol4", "sol#4", "la4", "la#4", "si4",
226  "do5", "do#5", "re5", "re#5", "mi5", "fa5", "fa#5", "sol5", "sol#5", "la5", "la#5", "si5",
227  "do6", "do#6", "re6", "re#6", "mi6", "fa6", "fa#6", "sol6", "sol#6", "la6", "la#6", "si6",
228  "do7", "do#7", "re7", "re#7", "mi7", "fa7", "fa#7", "sol7"]

```

Listagem 6.13 – Função notasCod

Assim, a estrutura dos dados para gerar o arquivo SMF desejado, conforme exemplo do capítulo 2, é como segue:

```

10 #musica = [ [96, // ppq
11             3, // numerador da formula de compasso
12             4, // denominador da formula de compasso
13             2, // armadura de clave
14             120], // metrônomo
15
16             [[1,1,1,1,1,1,2,2], // nota->1, acorde ->0 ou 2
17             [78,78,78,78,78,78,78,78], // instrumento [0..127]
18             [4,4,4,4,2,2,1,1], // figura [1..]
19             [60,62,67,64,63,500,63,67], // nota [0..127], pausa -> 500
20             [100,120,127,125,125,127,127,127], // volume [0..127]
21             [0] // canal [0..15]
22 ],
23 [0,0,0,1,1,1,1,1],
24 [1,1,1,1,1,1,1,1],
25 [2,2,1,1,4,4,4,4],
26 [69,52,57,54,53,50,65,62],
27 [100,120,127,125,125,127,127,127],
28 [6]
29 ],
30 [1,1,1,1,1,0,0,0],
31 [7,7,7,7,7,7,7,7],
32 [4,4,4,4,4,4,4,4],
33 [60,62,67,64,63,500,55,67],
34 [100,120,127,125,125,127,127,127],
35 [5]
36 ] ]
37

```

Listagem 6.14 – Estrutura dos dados no formato 1

Ao rodar um programa que aplica a função **geraMidiF1** nesta lista (*musica*), obtem-se o mesmo resultado obtido teoricamente, como se pode ver logo a seguir:

```
77 84 104 100 0 0 0 6 0 1 0 4 0 96 77 84 114 107 0 0 0 25 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88
4 3 2 24 8 0 255 47 0 77 84 114 107 0 0 0 31 0 192 24 0 144 69 100 96 128 69 0 96 144 69 100 0 144
72 100 96 128 69 0 0 128 72 0 0 255 47 0 77 84 114 107 0 0 0 23 0 193 73 96 145 67 100 48 129 67 0
0 145 64 100 48 129 64 0 0 255 47 0 77 84 114 107 0 0 0 16 0 193 73 0 145 62 100 129 64 129 62 0 0
255 47 0
```

Figura 6.17 – Resultado teórico

RESULTADO OBTIDO COM A FUNÇÃO **geraMidiF1**

(é mostrado como [Int] em vez de {Char} para que se possa comparar com o resultado teórico)

```
[77,84,104,100,0,0,0,6,0,1,0,4,0,96,77,84,114,107,0,0,0,25,0,255,88,4,3,2,24,8,0,255,89,2,1,0,0,255,81,
3,7,161,32,0,255,47,0,77,84,114,107,0,0,0,31,0,192,24,0,144,69,100,96,128,69,0,96,144,69,100,0,144,
72,100,96,128,69,0,0,128,72,0,0,255,47,0,77,84,114,107,0,0,0,23,0,193,73,96,145,67,100,48,129,67,0,
0,145,64,100,48,129,64,0,0,255,47,0,77,84,114,107,0,0,0,16,0,193,73,0,145,62,100,129,64,129,62,0,0,
255,47,0]
```

Figura 6.18 – Resultado obtido com a função geraMidiF1

 **Obs.** Observe que na função **geraMidiF1** já se corrigiu a ação para não gerar o evento de novo instrumento caso o instrumento anterior seja igual ao da presente nota.

A listagem 6.15 mostra o programa utilizado para rodar a função **geraMidiF1**, com o respectivo resultado obtido da aplicação da mesma nas listas de dados anteriormente descritas.